Universidade de Brasília – UnB
Faculdade do Gama – FGA
Curso: Engenharia de *Software*
Disciplina: Desenho de *Software*

Diagramas Estáticos

Versão 1.2

Attany Nathaly Lima Araújo **– 11/0147006**

Elaine Cristina Meirelles Peronico **– 12/0010551**

Tainara Santos Reis **– 10/0131280**

Vanessa de Andrade Soares - **12/0043190**

Brasília – DF, 19 de Junho de 2015

HISTÓRICO DE VERSÃO

a) Histórico de versão do documento

Versão	Data	Descrição	Autor
0.1	09/04/15	Criação do documento.	Tainara
0.2	11/04/15	Elaboração dos tópicos	Tainara
0.3	13/04/15	Revisão final	Tainara
1.0	13/04/15	Entrega do documento	Tainara
1.1	09/05/15	Estruturação dos tópicos a serem abordados	Tainara

b) Histórico de versão do modelo de domínio

Versão	Data	Descrição	Autor
0.1	09/04/15	Modelagem das classes conceituais	Tainara
0.2	10/04/15	Estabelecimento das associações e multiplicidades	Tainara
0.3	12/04/15	Aplicação de correções	Tainara
0.4	13/04/15	Inserção dos atributos e seus respectivos tipos	Tainara

c) Histórico de versão do 2iagram de classes

Versão	Data	Descrição	Autor
0.1	27/04/15	Evolução do modelo de domínio focada em adicionar as operações particulares de cada classe	Tainara
0.2	04/05/15	Estabelecimento das modalidades de associações e atualização das multiplicidades	Tainara
0.3	05/05/15	Validação do 2iagram com consulta a	Attany, Elaine,

		um especialista	Professora Milene, Vanessa, Tainara
0.4	09/05/15	Aplicação de padrões GRASP	Attany, Elaine, Tainara, Vanessa
0.5	10/05/15	Correções das associações e adição de operações advindas das associações	Tainara
0.6	11/05/15	Correção ortográfica (retirada de cedilhas e acentos).	Elaine
0.7	19/06/15	Atualização final do diagram de classes (padrões GoF)	Vanessa Soares

SUMÁRIO

1. INTRODUÇÃO 5

1.1 ESCOPO 5

2. MODELO DE DOMÍNIO 5

2.1 CLASSES CONCEITUAIS 5

2.2 CLASSES CONCEITUAIS 5

2.3 ASSOCIAÇÕES 7

2.4 MODELO DE DOMÍNIO 8

3. DIAGRAMA DE CLASSES 11

REFERÊNCIAS 17

1. INTRODUÇÃO

Este documento ter como finalidade a identificação das classes conceituais, seus atributos e associações tornando claro o vocabulário e os conceitos do sistema representados por intermédio do Modelo de Domínio. É apresentado igualmente o Diagrama de Classes que contempla uma representação mais robusta e complexa que o Modelo de Domínio. Seu foco está em tornar clara a forma como as relações entre as classes poderão ser implementadas, a visibilidade dos métodos e atributos, bem como classes críticas do sistema.

O potencial destes modelos reside em fornecer os insumos para decisões raciocinadas de implementação, desenvolvimento de estratégias que induzam à qualidade dos incrementos de software gerados.

1.1 Escopo

Este documento se aplica ao site Lovelace desenvolvido na disciplina Desenho de Software na Universidade de Brasília pelo curso de Engenharia de Software.

2. Modelo de Domínio

2.1 Classes conceituais

O modelo de domínio disposto neste documento se aplica ao cenário dos casos de uso:

- Manter Produto;
- Manter Categoria e SubCategoria;
- Realizar Cadastro;
- Concluir Compra;
- Manter Carrinho;
- Consultar Produto.

2.2 Classes conceituais

Foram identificadas treze classes conceituais importantes para o sistema. Elas são descritas na Tabela 1.

Tabela 1 – Lista de classes conceituais

id	Classe Conceitual	Descrição/Categoria de classe conceitual
1	*CarrinhoDeCompras*	Contêiner de itens.
2	*CatálogoDeProdutos*	Contempla todas as categorias de produtos.
3	*Categoria*	Agrupamento das subcategorias.
4	*Cliente*	Pessoa relacionada à transação.
5	*Pedido*	Transação de negócio. Evento da compra, aquisição de itens.
6	*DescriçãoDoProduto*	Registra o preço, o nome e as informações gerais do produto. *DescriçãoDoProduto* não representa um *Item*, mas uma descrição de informações sobre itens como sugere a UML.
7	*Item*	Produto relacionado à uma linha de item de transação. É um item físico da loja, possuidor de um número único que o identifica. Traz o conceito de ser passível de ser "colocado" em um contêiner que é o *CarrinhoDeCompras.*
8	*LinhaItem*	Transação de item de pedido.
9	*Loja*	Organização relacionada à transação.
10	*PagamentoPorBoletoBancário*	Instrumento financeiro que conduz ao pagamento por boleto bancário.
11	*PagamentoPorCartaoDeCrédito*	Instrumento financeiro que conduz ao

		pagamento por cartão de crédito.
12	*SistemaDeAutorizaçãoDeCrédito*	Sistema colaborador responsável por autorizar o pagamento por crédito.
13	*SubCategoria*	Agrupamento de itens.

2.3 Associações

Baseado nas associações sugeridas pela literatura (CRAIG, 2007) pode-se estabelecer as relações entre as classes conceituais conforme esclarece a Tabela 2.

Tabela 2 – Lista de associações

id	Categoria	Associações
1	**A** é uma transação relacionada à outra transação **B**	*PagamentoPorCartaoDeCrédito - Pedido* *PagamentoPorBoletoBancário - Pedido*
2	**A** é uma linha de item de uma transação **B**	*LinhaDeItem - Pedido*
3	**A** é um produto ou serviço para uma transação (ou linha de item) **B**	*Item - LinhaDeItemDePedido*
4	**A** é um papel relacionado a uma transação **B**	*Cliente - Pedido*
5	**A** está física ou logicamente contido em **B**	*CarrinhoDeCompras – Loja* *Item - Loja*
6	**A** é uma descrição para **B**	*DescriçãoDoProduto - CatálogoDeProdutos*
7	**A** é um membro de **B**	*Categoria - CatálogoDeProdutos* *SubCategoria – Categoria*

		Item – SubCategoria *CarrinhoDeCompras - Loja*
8	**A** usa ou gerencia ou possui **B**	*CatálogoDeProdutos – Loja* *PagamentoPorCartaoDeCrédito – SistemaDeAutorizaçãoDeCrédito*
9	**A** é conhecido/ registrado/ relatado ou capturado em **B**	*CarrinhoDeCompras - Compra*

2.4 Modelo de domínio

Com as associações estabelecidas, o próximo passo seguro foi partir para a modelagem em si. Foram modelados dois modelos principais distintos (Figuras 1 , 2, 3). A princípio foi realizada uma versão simplificada (Figuras 1 e 2). Ocultaram-se os atributos das classes visto que a literatura utilizada, conforme interpretaram os desenvolvedores deste projeto, sugere que sejam incluídos os atributos sempre que estes e os quais agregarem valor ao modelo e que satisfaçam aos requisitos de informação dos cenários em desenvolvimento. Ou seja, é necessário incluir um atributo sempre que este implica na necessidade de ser memorizado para os requisitos do cenário descrito (CRAIG, 2007; págs 182 e 183).

Para ambas modelagens apenas atributos simples como strings e números foram considerados por serem os tipos de atributos mais adequados para este contexto. Um atributo como *idCliente* é desconsiderado por não ser um tipo simples mas um tipo *Cliente.* Para isso e igualmente por esta razão o enfoque do modelo de domínio são as associações utilizadas.

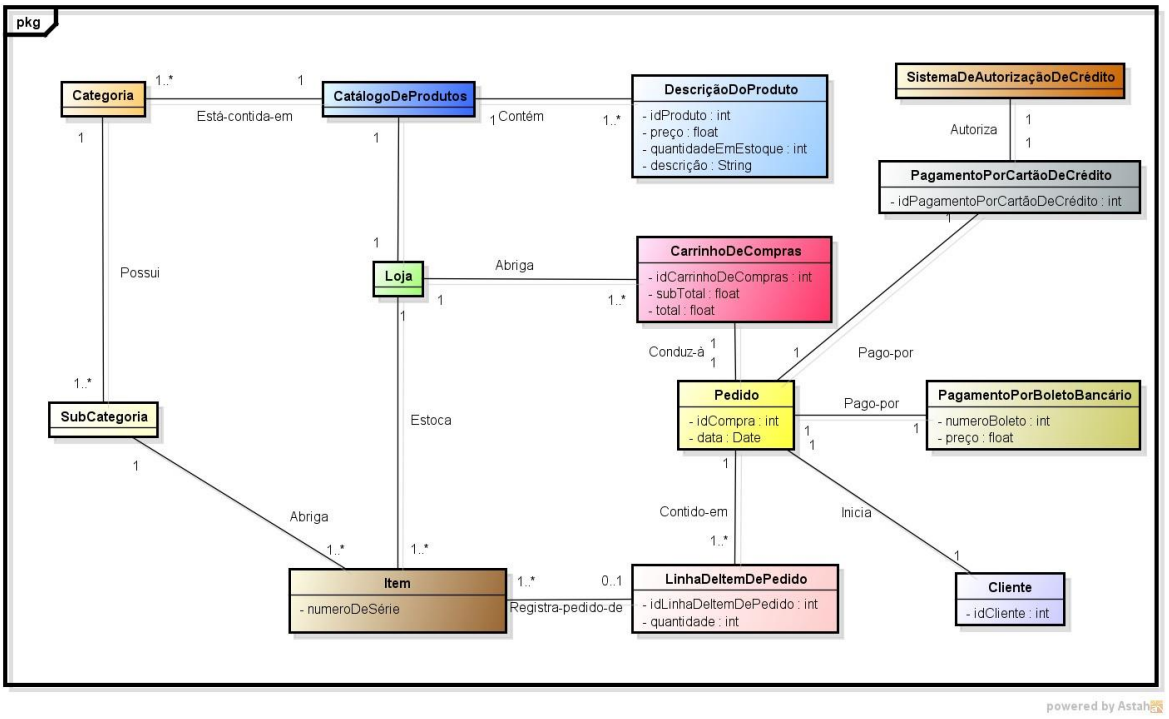

Figura 1 – Modelo de domínio simplificado. Versão 1.

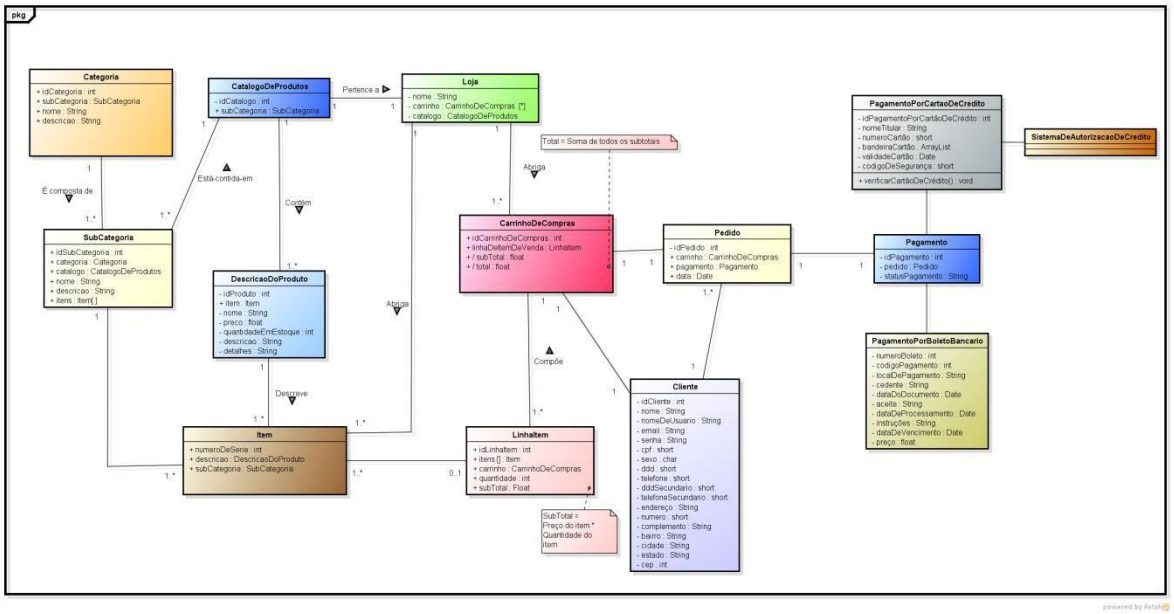

Figura 2 – Modelo de domínio simples. Versão 2.

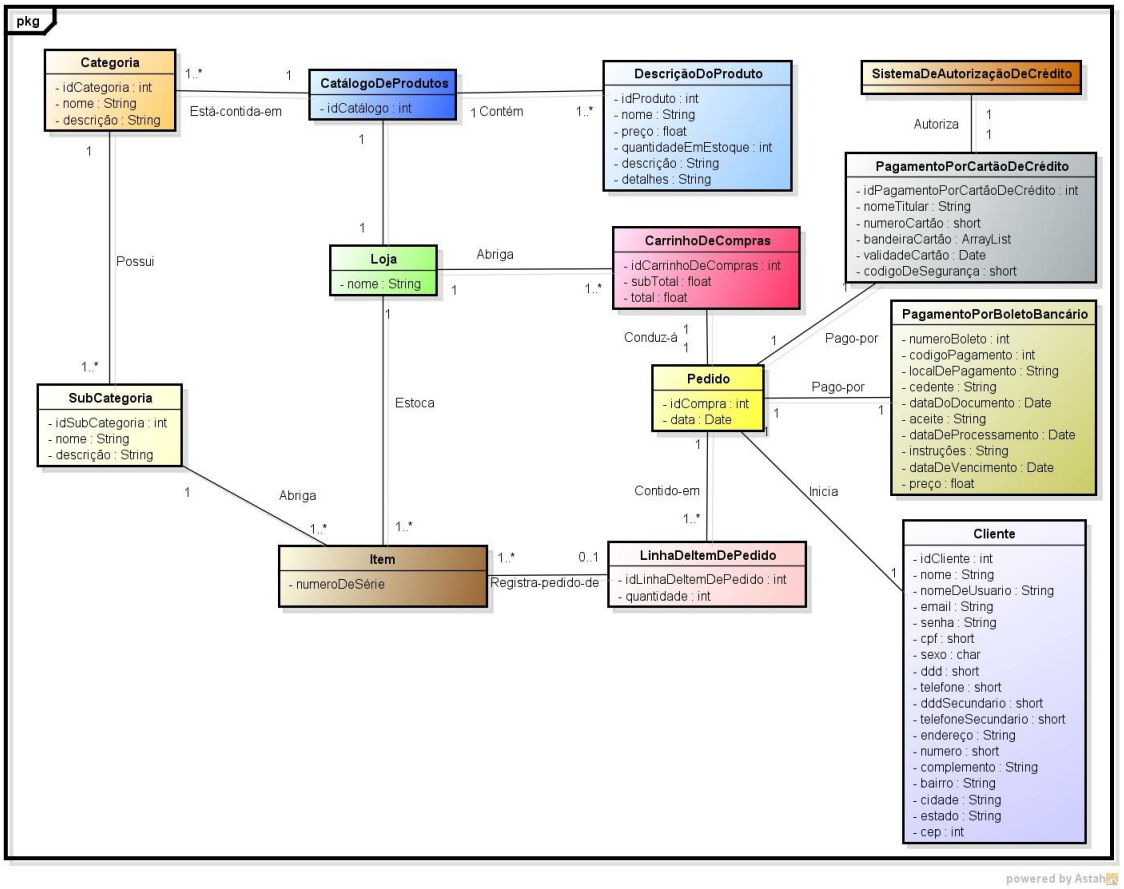

Figura 3 – Modelo de domínio completo. Versão 1.

3. Diagrama de Classes

Para o design do diagrama de classes foram aplicados princípios sugeridos pelo padrão GRASP. Segundo Craig:

> "Um modo popular de raciocinar sobre projetos de objetos de software e também sobre componentes em larga escala é em termos de responsabilidades, papéis e colaborações. Isso é parte de uma abordagem mais ampla chamada projeto guiado por responsabilidades ou PGR." (CRAIG, 2007, p. 292).

Um bom padrão identifica um problema e mostra a solução de modo a ser aplicado em diferentes contextos (CRAIG, 2007, p. 295). São nove os padrões GRASP como descreve a Tabela 3. Cada um deles foi analisado para se aplicar no projeto Lovelace.

Tabela 3 – Resumo dos padrões GRASP. Fonte: CRAIG, 2007.

Padrão GRASP	Problema	Solução	Associações
Criador (*Creator*)	*Quem cria A?*	✓ Atribuir à classe B a responsabilidade de criar uma instância da classe A se uma das seguintes afirmativas for verdadeira (quanto mais melhor): • B contém A, ou agrega A de forma composta; • B registra A; • B usa A de maneira muito próxima • B contém os dados iniciais de A.	→ *CarrinhoDeCompras* **cria** *LinhaItem;* → *Pedido* **cria** *Pagamento;* → *CatalogoDeProdutos* **cria** *Subcategoria;* → *Pedido* **cria** *Pagamento;*
Especialista na Informação (*Expert)*	*Qual é o princípio básico pelo qual atribuir responsabilidades a objetos?*	✓ Atribua responsabilidade à classe que tenha informação necessária para satisfazê-la.	→ *CatalogoDeProdutos* **é responsável por conhecer** *DescricaoDoProduto;* → *DescricaoDoProduto* **é responsável por conhecer** *Item.* A classe *DescriçãoDoProduto* armazena informações sobre itens. Ela não representa

			um *Item*, mas uma descrição de informações sobre itens; → *LinhaItem* **sabe** o subtotal da linha de item; → *DescricaoDoProduto* **sabe** o preço do produto; → *Pedido* **sabe** o total da venda.
Acoplamento Baixo (*Low Coupling*)	*Como reduzir o impacto de modificação?*	✓ Atribuir responsabilidades de modo que acoplamento (desnecessário) permaneça baixo. Use esse princípio para avaliar alternativas.	→ *Item* é uma classe com alto acoplamento pois está conectada a quatro outras classes com multiplicidade de 1 para muitos com todas elas. Em representações futuras, esta classe poderá ter suas conexões repensadas objetivando a diminuição do acoplamento.
Controlador (*Controller*)	*Qual é o primeiro objeto, além da camada de IU, que recebe e coordena ("controla" uma operação do sistema?*	✓ Atribuir responsabilidade a um objeto que representa uma dessas escolhas: • Representa todo o "sistema", um "objeto raiz", um dispositivo dentro do qual o software está sendo executado, ou um subsistema importante (todas essas são variações de um *controlador fachada*). • Representa um cenário de caso de uso	→ *CarrinhoDeCompras* **controla** várias **operações do sistema.**

		dentro do qual a operação do sistema ocorre (um caso de uso ou um *controlador de sessão*).	
Coesão Alta (*High Cohesion*)	*Como manter os objetos focados, inteligíveis e gerenciáveis e, como efeito colateral, apoiar Acoplamento Baixo?*	✓ Atribuir responsabilidades de modo que a coesão permaneça alta. Use isso para avaliar alternativas.	→ *Categoria, SubCategoria* poderiam ser apenas atributos de *Item.* Porém, visando a boa distribuição de foco para cada classe (coesão), elas precisaram ser criadas. → *LinhaItem* é necessária para evitar efeitos colaterais sob os objetos da classe *Item.* Ou seja, é uma alternativa que visa a alta coesão.
Polimorfismo	*Como tratar alternativas com base no tipo?* *Como criar componentes de software interconectáveis?*	✓ Quando alternativas ou comportamentos relacionados variam segundo o tipo (classe), atribua a responsabilidade pelo comportamento aos tipos para os quais o comportamento varia, usando operações polimórficas.	→ *Pagamento* **tem um comportamento default** com relação à *PagamentoPorBoletoBancario* e *PagamentoPorCartãoDeCrédito.* De modo que *PagamentoPorCartaoDeCredito* possui um comportamento que varia.
Indireção	*A quem devemos atribuir a responsabilidade de maneira a evitar o acoplamento direto entre dois (ou mais)*	✓ Atribuir a responsabilidade de ser o mediador entre outros componentes ou serviços a um objeto intermediário, para que eles não sejam diretamente acoplados.	✓ *LinhaDeItem* **é mediador** entre *Item* e *CarrinhoDeCompras;* ✓ *Pedido* **é mediador** entre *CarrinhoDeCompras* e *Pagamento;*

	objetos? Como desacoplar os objetos, de modo que o acoplamento baixo seja apoiado e o potencial de reuso permaneça mais alto?	O intermediário criar uma *indireção* entre os outros componentes.	✓ *DescricaoDoProduto* **é mediador** entre *CatalogoDeProdutos* e *Item*.
Variações Protegidas	*Como proteger objetos, subsistemas e sistemas de modo que as variações ou a instabilidade nesses elementos não tenham um impacto indesejável sobre outros elementos?*	✓ Identificar pontos de variação ou instabilidade previsível; ✓ Atribuir responsabilidades para criar uma interface (interface no sentido de comunicação) estável em torno deles.	❖ Para este projeto será adotado o encapsulamento (bem como polimofismo e a indireção) como mecanismo de variação protegida. Em próximas iterações os modificadores de acesso serão definidos e representados no diagrama de classes.
Invenção Pura	*Que objeto deve ter a responsabilidade quando não se quer violar a Coesão Alta e o Acoplamento Baixo ou outros objetivos, mas as soluções oferecidas pelo Especialista (por exemplo) não são apropriadas?*	✓ Atribuir um conjunto de responsabilidades altamente coeso a uma classe artificial ou de conveniência que não represente um conceito no domínio do problema – algo inventado, para apoiar coesão alta, acoplamento baixo e reuso.	❖ Não foi identificado para o contexto deste projeto.

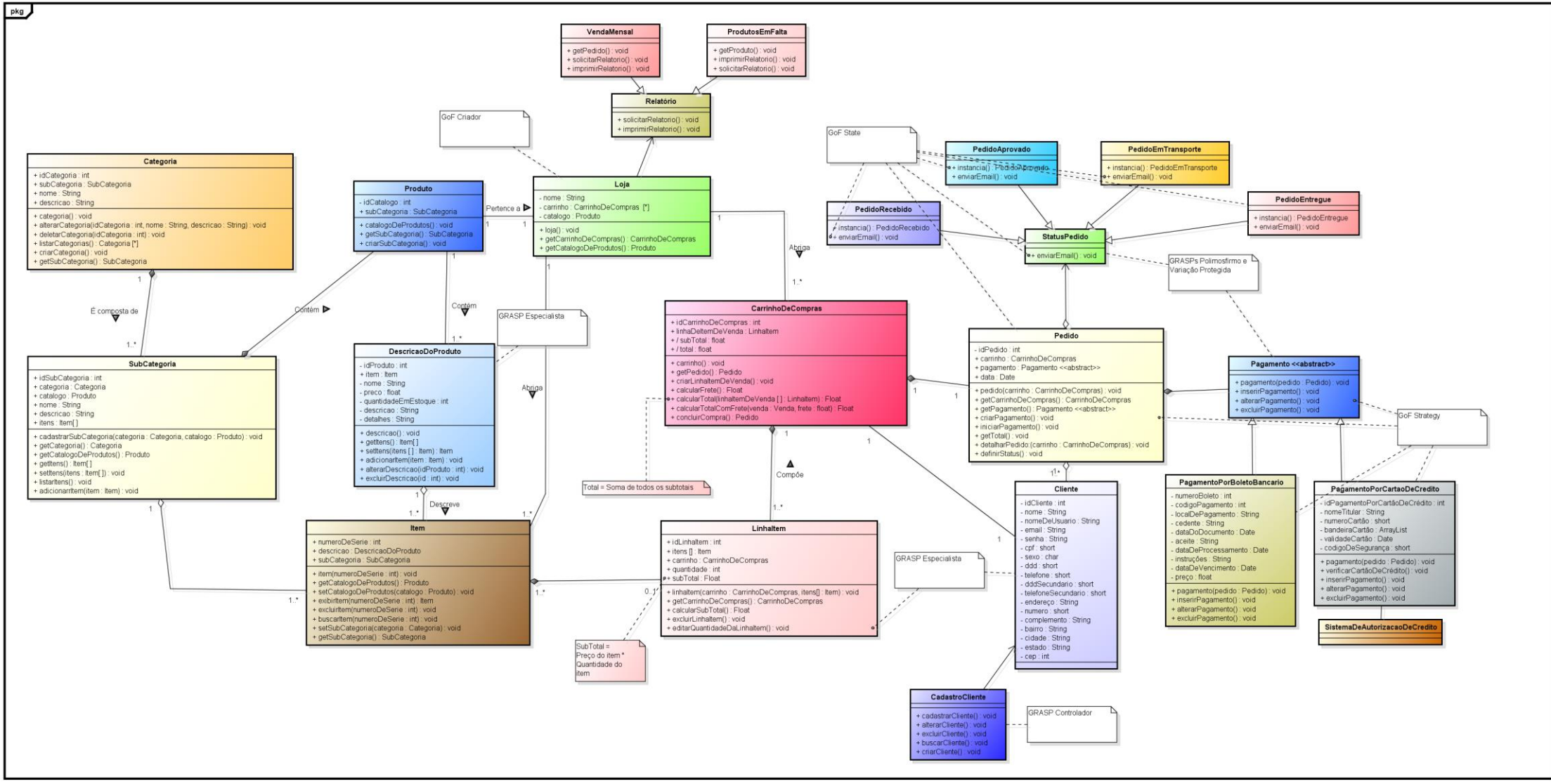

Figura 4. Diagrama de Classes. Versão 1.1.

REFERÊNCIAS

LARGMAN, CRAIG. **Utilizando UML e Padrões: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo.** Tradução Rosana Vaccare Braga ... [et al]. – 3. Ed . Bookman. Porto Alegre, 2007.